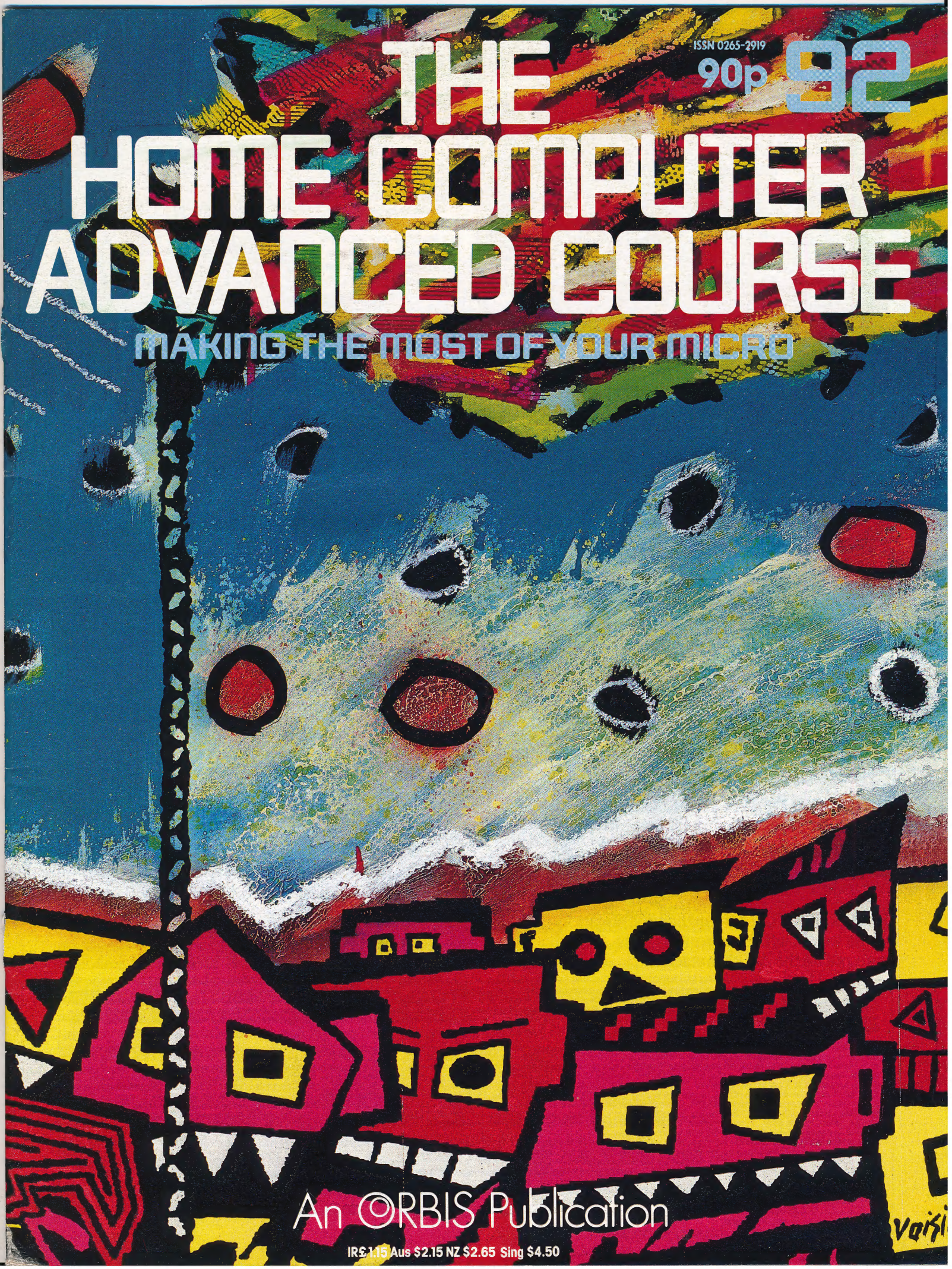
ISSN 0265-2919

90p 92

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO



An ORBIS Publication

IR£1.15 Aus $2.15 NZ $2.65 Sing $4.50

CONTENTS

APPLICATION

SELLING YOURSELF A look at computer-related careers in sales and marketing 1821

MACHELANGELO The artistic potential of the Macintosh is brilliantly displayed in the work of Frederic Voisin 1830

HARDWARE

IT'S FOR YOO-HOO! A modem can link your micro with a world of possibilities 1826

SOFTWARE

UNIX UTILITIES Unix offers a library of calls and commands to simplify user management 1824

GETTING IT OUT OF YOUR SYSTEM We end our series with a discussion of how to make function calls to MS-DOS from assembly language 1836

COMPUTER SCIENCE

ALL DIFFERENT TYPES It is possible to define your own data types in a C program 1832

JARGON

FROM UART TO VDU A weekly glossary of computing terms 1829

PROGRAMMING PROJECTS

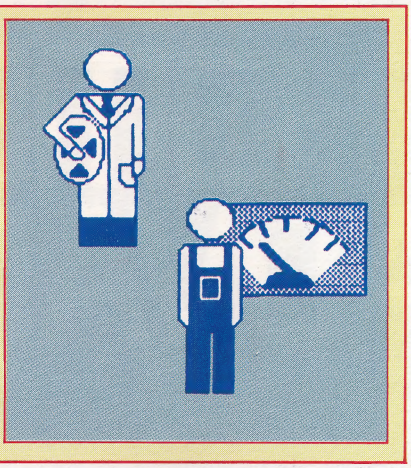
KNOCKED DOWN TO SIZE We look at three ways of making text files use up less space in memory 1834

MACHINE CODE

STACKS OF IDEAS Further investigation of the way in which subroutine calls manipulate the 68000's stacks 1838

Next Week

- **The Transputer has been described as a 'computer on a chip'. We investigate the potential and design of this revolutionary device.**
- **Functional programming is considered to be one of the solutions to the problems caused by normal sequential processing. We discuss the concepts behind functional programming languages.**
- **Our series on careers in computing continues with a look at the jobs of the technician and engineer.**



QUIZ

1) Where might you find the Hayes standard?
2) What purpose is performed by the 'link' commands in 68000 assembly language?
3) What are 'clusters'?
4) What kind of information is likely to be contained in a signal pattern?

Answers To Last Week's Quiz

1) AmigaDOS is based on the Tripos operating system.
2) The EPROM banks out the BASIC ROM in memory occupying that address space. This allows an additional 16 Kbytes of program to be included, which otherwise would not be available.
3) A pipe is a utility that allows a number of commands to be strung together, with the output from one command becoming the input for the next.

Coming Up

- A discussion of some of the developments expected in the microcomputer industry in forthcoming years.
- A look at the things to consider when choosing a computer language.

Editor Stephen Cooke; **Art Editor** Claudia Zeff; **Deputy Editor** Steve Colwill; **Production Editor** Bobby Pickering; **Designer** Julian Dorr; **Staff Writer** Steve Malone; **Art Assistant** Caroline Clayton; **Sub Editor** Jon Kaye; **Contributors** Mike Curtis, Steve Malone, Nigel Cross, David Fensome, Dennis Freeman, Claudia Zeff, Dave Nicholls, Peter Shaw; **Software Consultants** Pilot Software City; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Maurice Geller; **Production Assistant** Susan Brown; **Subscription Manager** Christine Allen; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 14 Rathbone Place, London W1P 1DE; **© APSIF Copenhagen 1985; © Orbis Publishing Ltd 1985: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Heanor Gate Printing Ltd, Derby**

HOW TO OBTAIN ISSUES AND BINDERS FOR THE HOME COMPUTER ADVANCED COURSE – Issues can be obtained by placing an order with your newsagent or direct from our subscription department. If you have any difficulty obtaining any back issues from your newsagent, please write to us stating the issue(s) required and enclosing a cheque for the cover price of the issue(s). **AUSTRALIA** – please write to: Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G, Melbourne, Victoria 3001. **MALTA, NEW ZEALAND & SOUTH AFRICA** – Back numbers are available at cover price from your newsagent. In case of difficulty, write to the address given for binders.
UK/EIRE – Issue Price: 90p/IR£1.15. Subscription: 6 months: £26.00. 1 Year: £52.00. Binder: please send £3.95 per binder, or take advantage of our special offer in early issues. **EUROPE** – Issue Price: 90p. Subscription: 6 months air: £44.72. Surface: £36.14. 1 year air: £89.44. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **MALTA** – Obtain binders from your newsagent or Miller (Malta) Ltd, MA Vassalli Street, Valetta, Malta. Price: £3.95. **MIDDLE EAST** – Issue Price: 90p. Subscription: 6 months air: £50.18. Surface: £36.14. 1 year air: £100.36. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **AMERICAS/ASIA/AFRICA** – Issue Price: US/CAN$1.95/90p. Subscription: 6 months air: £59.54. Surface: £36.14. 1 year air: £119.08. Surface: £72.28. Binder: £5.00. Airmail: £9.50. **SOUTH AFRICA** – Issue Price: SA R2.45. Obtain binders from any branch of Central News Agency or Intermag, PO Box 57394, Springfield 2137. **SINGAPORE** – Issue Price: Sing $4.50. Obtain binders from MPH Distributors, 601 Sims Drive, 03-07-21, Singapore 1438. **AUSTRALASIA/FAR EAST** – Issue Price: 90p. Subscription: 6 months air: £64.22. Surface: £36.14. 1 year air £128.44. Surface: £72.28. Binder: £5.00. Airmail: £9.75. **AUSTRALIA** – Issue Price: Aus$2.15. Obtain binders from First Post Pty Ltd, Locked Bag No. 1, Cremorne, NSW 2090. **NEW ZEALAND** – Issue Price: NZ$2.65. Obtain binders from your newsagent or Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington.
ADDRESS FOR BINDERS AND BACK ISSUES – Orbis Publishing Limited, Orbis House, Bedfordbury, London WC2 4BT. Telephone 01-379 5211. Cheques/postal orders should be made payable to Orbis Publishing Limited. Binder prices include postage and packing and prices are in stirling. Back issues are sold at the cover price, and we do not charge carriage in the UK.
NOTE – Binders and back issues are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK and Australian markets only. Binders and Issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.
ADDRESS FOR SUBSCRIPTIONS – Orbis Publishing Limited, Hurst Farm, Baydon Road, Lambourn Woodlands, Newbury Berks, RG16 7TW. Telephone: 0488-72666. All cheques/postal orders should be made payable to Orbis Publishing Limited, Postage and packaging is included in subscription rates, and prices are given in sterling.

PAINTING ON THE COVER BY FREDERIC VOISIN

SELLING YOURSELF

MARCUS WILSON-SMITH

Setting Up Shop
For many untrained job candidates looking for a career in sales or marketing, probably the best place to start is selling equipment to end-users in an outlet such as this one. However, prospects remain slim in an environment that may be too pressurised to allow the employee to gain more than a superficial awareness of new technology and sales techniques

In an increasingly sophisticated marketplace, the emergence of sales and marketing as essential components of computer companies has created many job and training opportunities. Careers in these two fields can, as we'll see, lead to both lucrative and high-ranking careers, whether with small companies or corporate giants.

The microcomputer industry's massive growth in the early 1980s began to level off in mid-1984 and job opportunities accordingly became tighter. Hundreds of companies have gone out of business but almost three times as many have been founded, indicating that plenty of new jobs are being created in computing. But even the best programmer or salesman can find himself out of a job overnight if his company closes down.

Jobs in sales and marketing are among the most well paid in the computer industry. At the top end, experienced salesmen negotiating valuable contracts for hardware or software with large corporate buyers can expect to make £80,000 or more a year, mainly in commissions. Marketing executives — who are not paid commission — are unlikely to be paid nearly as well, although by the time an employee reaches the position of marketing director in a medium to large company, salaries in excess of £30,000 are quite common. Both sales and marketing are well-trodden routes by which 'high-flyers' rise to the very top of a company. Until recently, all IBM chief executives had risen through the sales ranks.

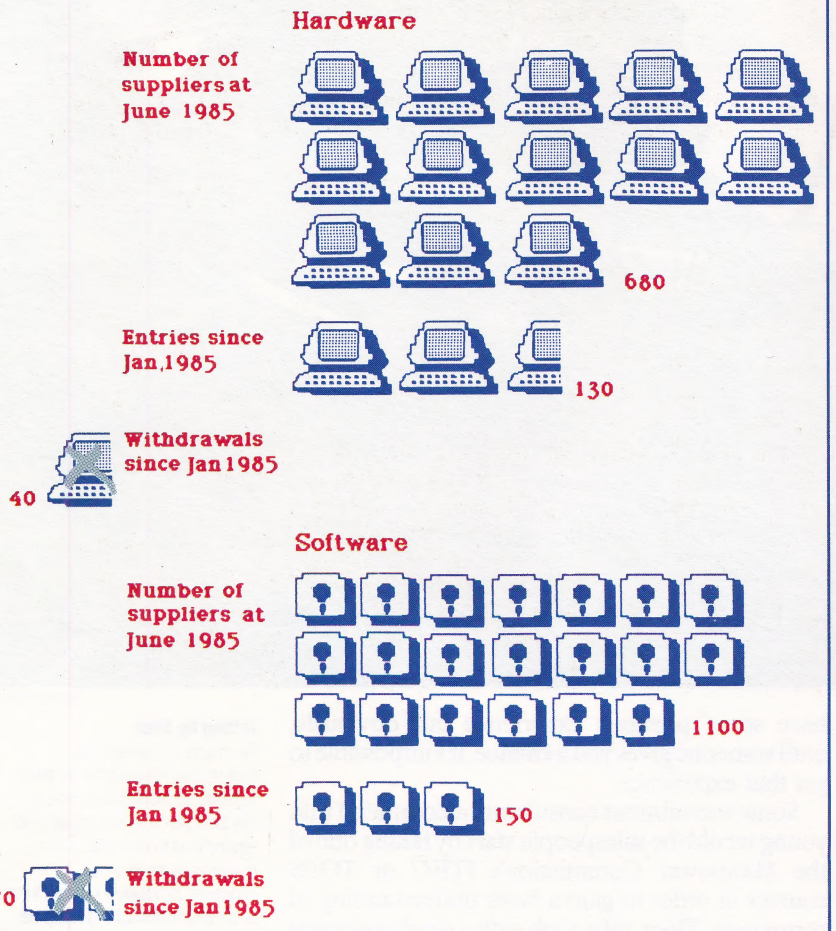
In most careers, the hardest part is getting onto the first rung of the ladder; sales is no exception. Companies normally require job applicants to have some previous experience but obviously, until someone gives you a chance, it's impossible to get that experience.

Some recruitment consultants recommend that young would-be salespeople start by taking one of the Manpower Commission's ITEC or TOPS courses in order to gain a basic understanding of computers. Then, take a job with a retail operation that sells office equipment or home computers. This will be quite badly paid —£80 per week is typical — but it will provide you with some basic experience. After that, you will be in a stronger position to persuade a manufacturing company or professional computer dealer to take you on. For the first year there you still won't be paid much — £5,000 or so per year — and you will be expected to follow a more senior salesman around and learn how to demonstrate particular computers and software packages.

After that, if you're any good, your prospects will improve very rapidly. Armed with one year's experience on any of the major micros and software packages (IBM, Apple, Apricot, Lotus and so on), you'll be in line for a sales executive job paying £8,000 to £10,000 basic, which can be doubled by commissions. You would very likely also get a company car. When a job is advertised as '£20,000 OTE', it means that £20,000 is the 'on-target earnings' you will receive only if you achieve the sales goal the sales manager has set for you. (According to recruitment specialists, most companies seem to be biased towards applicants who are in their early or mid-twenties.)

Within three or four years, you can expect to become a senior sales executive or systems salesman, handling larger accounts. In a company that sells both single-user and multi-user

Growth Of Supplies



CAROLINE CLAYTON

Computer Supplies
Despite the pessimistic forecasts of some observers, the number of suppliers within the computer industry has continued to grow steadily. Stable growth, rather than excessive profits, will often tempt new businesses into a market which they might otherwise avoid (these figures from NCC Microsystems Centre)

computers, the senior staff would naturally handle the more expensive, multi-user systems that net higher commissions. Your earnings at this point could be £15,000 basic, £30,000 OTE. By the age of 30, you will be looking for a sales manager position, where you're the one setting the sales targets for your sales staff. At this managerial level, you may find your job overlapping the marketing manager's and many people in fact switch over to marketing at this point.

Keep in mind, though, that sales can be a vicious business, especially in the high-risk micro industry. Many salespeople have found they have been set unattainable targets by their sales manager. After failing to reach their targets for two or three quarters, they find themselves summarily dismissed. This is not a job for someone who values security!

If you find that you don't like sales, or that you're not particularly good at it, the best advice is to get out while you are young. If you reach your mid-thirties and have not yet become a sales manager, you may find it very difficult to find another job in sales.

THE MARKETING SIDE

Marketing is a highly competitive field. Although some companies will accept a Higher National Diploma (HND) qualification, most firms are looking for university graduates, preferably with a good honours degree in a numerate discipline like maths or physics.

An important point to realise about marketing is that the same principles apply no matter what product your company sells. So experience gained in the marketing department of an organisation that produces soap powder will stand you in good stead when you come to market computers. In fact, many consultants recommend that young graduates get two or three years under their belts working for a large blue chip corporation, preferably one that will send you on Institute of Marketing courses or pay for a part-time Master of Business Administration (MBA) degree. Computer firms tend to be smaller with fewer resources to train their employees or allow them time off to take outside courses.

Knowledge of computers is not vital in getting a marketing job in a microcomputer firm, but obviously an interest in the subject helps. A marketing assistant with one or two years experience will earn in the region of £10,000 per year. The sort of tasks you would be doing for a microcomputer manufacturer include writing brochures and advertising copy, sending information to the dealer network, organising seminars and press conferences, and dealing with journalists (normally passing them on to someone

The Sell Game

The sales and marketing functions in a company are often confused. A salesman deals directly with the customer. The marketing executive, on the other hand, is involved more with strategy: how to package the product, what to call it, how to advertise it, and so on.

There are two main branches in the microcomputer industry — manufacturing and reselling. Sales people working for hardware and software manufacturers typically sell their products to dealers or distributors. Salesmen for resellers, such as systems houses, dealers or retail chains, go directly to the end-user. Unlike in the USA, where it's common for customers to walk in off the street into a computer shop to buy a business computer, almost all micro sales in the UK involve a direct mailshot to potential customers (a marketing function) or 'cold calling' (where the salesman makes a direct approach by telephone or in person).

In the early days of microcomputing, the industry did not pay much attention to marketing. Fledgling companies made exciting new products and salesmen sold them. Sometimes they were successful (the Sinclair Spectrum), but sometimes they weren't (the Coleco Adam).

The industry is now more sophisticated, though, and the most successful home computer manufacturer of 1985 was Amstrad, with its CPC series. There was nothing new about the technology; it was simply a matter of packaging it in the way that customers wanted. That, fundamentally, is what's meant by marketing

Chain Reaction

Granada Business Centres is a growing chain of microcomputer dealers, selling single- and multi-user micros to small and medium sized businesses. The company says the prime requirement of its sales recruits is that they have sales experience, preferably in selling business equipment, or (even better), selling computer systems. Granada runs its own training on products and sales skills, with most sales staff being hired between the ages of 23 and 30. Each store has a senior sales executive selling into large 'national accounts', and a sales manager who supervises the sales activity of his store. Then there is a district manager who looks after a group of four centres including the maintenance and support staff who serve each group. The career path, then, is as follows:

salesman → senior sales executive → sales manager → district manager

Granada's marketing staff is much smaller: just four people, compared to 60 on the sales side. Each marketing manager tends to have a specific function. One person looks after exhibitions, another deals with the press and PR, another with the appearance of the stores and so on

more senior at this stage).

The next step up is marketing manager, where you would take a more direct role in liaising with the advertising agency, deciding which exhibitions to attend, planning product launches and company strategy. Your salary now would be £14,000 and upwards with a company car and other 'perks' thrown in. In a small company — as most microcomputer firms are — you could shortly expect to be made marketing director with a seat on the board.

Small Wonder

Psion is a British manufacturer producing software and hand-held computers. It employs 70 people and expects its sales recruits to have had some experience selling technical or office equipment, but does not insist on knowledge of computers. Internal training is given on the company's products. The salesman will be dealing directly with the dealers who handle Psion products and large corporate buyers. A typical career path would be:

salesman → account manager → sales manager → sales director

An account manager is responsible for handling a number of accounts and supervising the sales force.

On the marketing side, Psion employs only graduates as marketing executives. Their job is to write copy for brochures and advertising, organise exhibition stands, handle queries from journalists and provide back-up for the sales force. The career path here is:

marketing executive → marketing manager → marketing director

The marketing manager and director are responsible for long-term planning, consulting with and instructing the advertising agency, commissioning market research and so on.

Small companies can offer advantages to an employee who isn't sure which career path to follow; they may have a much more flexible structure than larger companies. In 1983, Psion took on a graduate as a programmer who then switched to technical sales support (answering technical queries the sales team couldn't handle themselves) before becoming a sales account manager. By the end of 1985 he had moved to Psion's export division

On The Market

Marketing represents far more than just selling a product. A marketing executive will be involved with a product from the conception and design stage right through to release, ensuring that the requirements and preferences of the intended end-user are not overlooked during product development. The Amstrad PCW 8256 is a good example of a well-marketed product. Although it uses yesterday's technology, it answers the needs of today's consumer for low-cost word processing. Compare the success of this product with, say, the Sinclair C5 electric vehicle, which failed to sell in volume despite intensive advertising — thus proving that no amount of publicity can sell a product that cannot find a market

COURTESY OF AMSTRAD

UNIX UTILITIES!

As a multi-user system, Unix is required to have a wide variety of useful and friendly tools and utilities for easy user management. Based for the most part on the C language, the utilities available on Unix are very straightforward library calls and commands, many of which we look at in this instalment.

Because Unix utilities and tools are straightforward C programs, it is easy to combine your own programs with them. To simplify matters even more, Unix has a fundamental unified approach to files and I/O devices. Every file is considered to be a stream or sequence of bytes. Unix will read or write bytes sequentially to or from the file and allow you to position the file pointer to any legitimate byte offset in the file and begin reading or writing at this point.

I/O devices are treated exactly like any other files. In fact, there will always be a list of I/O devices in the /dev directory. The operations that can be performed are best looked at using the lowest level C library calls:

- creat(filename,protectionmode): If a file with the given name already exists, then it is truncated to zero length; otherwise, a new file with that name is created. The protectionmode is a nine-bit number (commonly in the form of three octal digits) specifying read, write and execute permission (one bit each) for the user, the user's group and everybody else on the system. The function returns an integer filedescriptor.
- open(filename,readwritemode): Connects the file named to the program, specifying (in readwritemode) 0 for read, 1 for write and 2 for both read and write access. The file must exist to start with and the function returns an integer filedescriptor.
- close(filename): Disconnects the file from the program.
- unlink(filename): Removes the file from the system.
- read(filedescriptor,buffer,numberofbytes) and write(filedescriptor,buffer,numberofbytes): Both of these transfer the stated number of bytes between the file, whose descriptor is given, and a named buffer area of store.

There are three files automatically opened for any program, which are called standardinput, standardoutput and standarderror. Normally, standardinput is connected to the keyboard and standardoutput and standarderror are connected to the screen. The first two can be redirected to or from any other file or device by means of the < and > operators.

For any Unix command or program:

```
commandname < fileordevicename
```

will cause input from standardinput to be actually taken from the named file rather than the keyboard. Similarly:

```
commandname > fileordevicename
```

will cause output to standardoutput to be sent to the named file rather than the screen. A variation will allow the output to be appended to a file rather than to create a new file, though a new file will be created if it does not exist:

```
commandname >> fileordevicename
```

The actions of these last two may be altered if noclobber is set (a customising option we'll be looking at in greater detail later). This will not allow the deletion of files without confirmation, but can be overruled with the imperative form of these two operators:

```
commandname >! fileordevicename
commandname >> ! fileordevicename
```

As an example of the use of this facility, we can create a file containing the list of files in a directory by using:

```
ls> lsfile
```

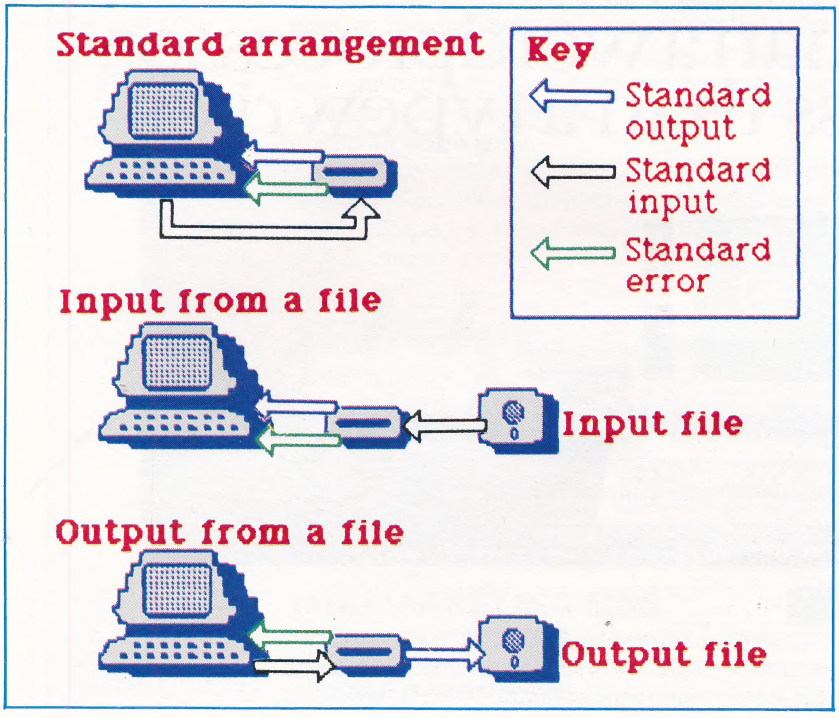
another use is to provide a command file facility, so that a sequence of commands can be executed all at once. Commands are executed by the 'shell', which can be invoked using the sh command. So:

```
sh < filename
```

will cause commands in the file to be executed.

One major feature of Unix that gives it so much power is the pipelining facility given by the

Redirection Routes
In normal circumstances, input to a program that is running on a Unix system is from the keyboard. Output and error reports are normally directed to the screen. The < and > operators in Unix enable either the input to, or the output from, a program to be redirected so that the program communicates with other devices or files. This powerful facility allows data files to be created simply, or a series of Unix commands held in a file to be executed by the 'shell'



CAROLINE CLAYTON

operator |. This is in the format:

```
commandorprogramname | commandorprogramname
```

which connects standardoutput from the first command to standardinput for the second. Thus output from the first command is automatically taken as input to the second.

This doesn't mean that an intermediate file is produced, but that the two programs will run concurrently. Whenever the second program needs input it will suspend operation until the first program has produced some output. For example, the ls command doesn't count the number of files in the directory as it lists them.

There is a utility, wc, which counts the number of words in a file using the option -w; the command:

```
ls | wc -w
```

causes the output from ls — a list of file names — to be the input to wc which counts the number of words, so the output will be the number of files in the directory.

One particular use of the pipeline operator is in conjunction with the tee command, which will send output to both a file and to the screen. So:

```
ls | tee lsfile
```

will cause the directory listing to appear on the screen as well as in lsfile.

Unix is a multi-tasking operating system and can therefore run two or more tasks concurrently. But Unix also allows each user to run more than one program at a time. If the & operator is placed at the end of the command line, the command or program will be executed in the 'background', leaving the user free to run another command or program. Of course, if this background process uses terminal input or output, then strange things are likely to happen.

It's also possible to stop a task in the middle of execution by pressing Ctrl-Z, which allows the task to be restarted later, without losing it. If you have a number of processes running at the same time and some stop, it can be difficult to know what is going on. The command ps, however, gives a list of the statuses of all processes currently running for the user, and the command jobs lists all background and stopped jobs.

When a process is stopped or put into the background, it is given two numbers by Unix — a personal job number such as [1], [2] and so on, and a process number that relates to all processes currently running on the machine. The job number can be used in conjunction with the fg and bg commands to swap processes about. The command fg %jobnumber, for example, will restart a stopped process in the foreground, or switch a background process to the foreground. The command bg %jobnumber will restart a process in the background, and a process running in the background can be stopped using stop %jobnumber. Processes no longer needed can be stopped permanently by the command kill processnumber.

Unix, like all good operating systems, maintains a time and date that can be used in a variety of ways. The command date simply gives you the day's date and cal gives you a calendar, which can be in a variety of formats. The command times commandname will tell you the execution time for that particular command and at followed by a time and a filename will execute a list of commands held in the file at the given time.

Directing The Traffic

```
%ls -l > lsfile
(no directory listing appears. It has gone to lsfile instead of to
the screen)

%cat lsfile          (look in lsfile)
total 41
-rw-r--r--          1 com-mcc       0  Oct 28  11:55 lsfile
drwxr-xr-x          2 com-mcc     512  Oct 21  11:11 mike
-rw-rw-r--          1 com-mcc     502  Sep 17  12:07 rec.c
-rwxr-xr-x          1 com-mcc   18432  Oct 21  11:02 receive
-rw-r--r--          1 com-mcc    1068  Oct 18  14:44 rx.p
-rwxr-xr-x          1 com-mcc   19456  Oct 21  11:03 transmit

%ls | sort -r > lsfile
(this time the directory list has been piped to the sort utility
with the option -r which sorts into reverse order)

%cat lsfile  (list sorted file)
transmit
rx.p
receive
rec.c
mike
lsfile

%pc rx.p &    (run pascal compilation in background)
[1] 1217      (compilation given job no. (1) and process no. 1217)

%jobs
[1] + Running  pc rx.p

%ps
PID   TT  STAT TIME COMMAND
1217 n09 I     0:00  pc rx.p
1218 n09 R     0:03  pc0 -o/tmp/p0001217 rx.p
(this last process was set up by the pascal compiler)
1220 n09 R     0:01  ps

%stop 1217          (compilation stopped by quoting process no.)
[1] +Stopped (signal)  pc  rx.p

%bg %1              (restart compilation in background using job
no.)
[1] pc  rx.p &

%fg %1              (move compilation into foreground)
pc rx.p

Stopped             (this was done using Ctrl Z)

%jobs
[1] +Stopped  pc rx.p

%kill %1            (get rid of the process completely)
[1] Exit 1  pc rx.p

%ps
PID   TT  STAT  TIME COMMAND
1266 n09 R     0:01  ps
(no processes running except ps itself)

%logout
```

IT'S FOR YOO-HOO!

Although it has been around for many years as an integral component of the telecommunications industry, the modem is only now beginning to make an impact in the UK, in both business and private sectors. The home micro user has several modems to choose from, five of which we examine here.

In the USA, modems are big business. In the UK, however, their impact has been less pronounced. There are a number of reasons for this, but they boil down to two major factors. First, the connection charges on the telephone network, particularly during peak hours, can be prohibitively high. People using their computers on the network for several hours a week are often stunned by their quarterly telephone bill, which can run into hundreds of pounds.

Secondly, British Telecom has been slow to realise the potential of communication networks and has developed its system in a piecemeal fashion. This means that the Prestel database, for instance, uses different protocols from private bulletin boards and so on. A further obstacle to development in the UK is that before a modem can be legally used on the telephone network it must first receive British Telecom approval — BT, however, has been notoriously slow in granting this. This has meant that in many cases several years have elapsed before advances in modem technology have filtered through to the user. Since the privatisation of BT, the authorisation of modems has been transferred to the Department of Trade and Industry, which should make the process more efficient.

In the USA, many local calls are free and almost all database and communications systems are based around the Hayes standard. The Hayes standard is a system of protocols which have been adopted by a number of other modem manufacturers. This standardisation is not available to the UK user, although development of the Hayes system in this country is imminent.

Thus in order to be fully adaptable in the UK, a modem requires these and a number of other features built in to allow the user maximum flexibility in the often confusing communications jungle (for further details see page 941). Despite these problems, the sale of modems continues to increase. Let's now look at the features offered by some of the modems on the market.

CRISPIN THOMAS

PACE NIGHTINGALE

PRICE
£137 excluding software and interface

FOR USE WITH
BBC Micro, Commodore 64 and Amstrad CPC range

BAUD RATE
Both Send and Receive can be set to one of seven baud rates from 75 to 9800

NIGHTINGALE

The Pace Nightingale is aimed specifically at the BBC Micro, although it can be adopted to any micro with an RS232 port. Like the Modem 1000 package, it provides the software on an EPROM that is inserted into one of the BBC Micro's sideways ROM sockets. Although several of the modem's features are contained within the menu-driven software, the Nightingale lets you set the baud rate, via buttons on the front of the device, for the more common Prestel and 300-baud bulletin board standards.

The software adapts the computer to be run in Prestel or terminal emulation modes. From either of these menus it is possible to transfer information and manipulate it within the computer's buffer.

The Nightingale also comes with the most comprehensive manual provided with any of the modems discussed here. Because of the large number of BBC Micros installed in educational establishments, the manufacturers have probably designed the manual with the education market in mind. But whatever the reason, it tells the user far more than the scant information provided with a number of other modems intended for home micros.

VTX 5000

Despite having a chequered history, the VTX 5000 (see page 426) manufactured by OE Ltd has become the standard modem for Spectrum owners. It was originally developed for Prism, which was interested in expanding the user base for Micronet in which the company had a major stake. Although the modem was taken over by Modem House when Prism crashed at the beginning of 1985, it has continued to sell well.

At the front of the modem is a power light, a Line light (indicating whether the modem is 'on-line') together with a switch. Another switch allows you to select the mode required for the device, which can be for Micronet, TX (transmit) or RX (receive). The back of the device holds a BT telephone jack socket and lead, which can be plugged directly into the wall.

The VTX 5000 has its software contained on ROM. Inside the VTX 5000 are two printed circuit boards, one that deals with the serial transmission and decoding circuitry while the other contains the Spectrum-specific logic such as the power feed ACIA chip and ROM-based software. This method of providing the modem software presents the Spectrum owner with a problem. The ROM works by paging out the BASIC ROM and occupying that area of memory. The trouble arises when users want to use the Interface 1 (see page 1398) along with the VTX 5000 — to download programs onto a microdrive, for example. Because the Interface 1 uses the same technique to page in its own ROM, the two devices cannot be used together.

VTX 5000
PRICE
£69.95
FOR USE WITH
Sinclair Spectrum
BAUD RATE
75/1200

The modem software is menu-driven and on power-up presents the user with a menu of options, including logging on to Micronet or sending messages via the mailbox facility. However, the software does not allow you to log on to any of the multitude of private bulletin boards that are springing up around the country, although it is possible to write your own software to do so.

MODEM 1000

Coming from the same stable as the VTX 5000, the Modem 1000 has many features in common with the dedicated Spectrum machine. However, the Modem 1000 is designed to be run with a wide variety of computers and so has no machine-specific ROMs on board.

The external controls are identical to those provided on the VTX 5000. Thus there is a Line switch and LED, and a three-way mode switch. The rear of the device contains a Data In port that provides the interface to the computer, as well as a telephone jack. Because the Modem 1000 is not powered from the computer, the device has been fitted with its own power lead and transformer.

The serial transmission circuitry and ACIA chip is almost identical to that used on the VTX 5000. However, in order for the user to communicate through the system, some software is required. For the BBC Micro, this is provided on ROM which can be fitted into one of the free sideways ROM sockets inside the computer. This provides features very similar to those on the Spectrum. Menu-driven programs allow you to download programs, log on to Micronet and receive and transmit messages.

Because of the nature of the Commodore 64, the interface required for the Modem 1000 is more complex than that for the VTX 5000. This is provided by a plug interface board that fits into the cartridge port of the computer. This interface contains not only the software that will page into memory instead of the computer's BASIC ROM, but also special adaptor circuitry that programs the Commodore ASCII set into the generally used communications standard. Another part of the board is dedicated to producing a standard RS232 signal that can be sent to the data port on the modem.

MODEM 1000
PRICE
£69.95 excluding software and interface
FOR USE WITH
All popular home micros
BAUD RATE
75/1200

COMMUNICATOR 104

Manufactured by KDS Electronics, this modem is designed for use only with the Amstrad CPC range of machines. Like the VTX 5000, this plugs directly into the expansion bus at the rear of the computer. But in this case, power is supplied from an external transformer fitted onto the power socket. Also, like the other modems we have looked at, this one is connected in serial with the telephone. So, in order to get on-line to a bulletin board or Prestel, you first dial the number on the telephone and then switch in the modem.

The Communicator differs from the other modems shown here in that it has two seven-segment displays on the front that provide you with information such as the number currently being dialled, and whether the modem's ROM is logged into the Amstrad system.

The software for the Communicator is installed on ROM within the modem itself. Like the OEL range, the programs are menu-driven, although the number of options available is far greater. The Mode Select menu effectively enables you to select which type of system you are intending to use — Prestel or bulletin board, for instance. Once the mode has been selected, the modem will automatically set the baud rate, although it is possible for you to choose your own. The menu-driven software also supports facilities for auto-dial, auto-answer and carrier detect.

The Communicator supports a large number of useful resident system extension commands. These include, among other things, commands to enter the modem's ROM, set the baud rate and data word options, and configure the printer to provide hard copy of the session. The Communicator's main drawback is that the normal telephone cannot be connected while the modem is in use. It's therefore necessary to physically plug the modem or telephone in the wall socket in order to select which piece of equipment you want to use.

CRISPIN THOMAS

WS3000

Marketed by Miracle Technology, this range of modems is at the top end of the microcomputer spectrum, containing the full range of features required by most users. It would therefore be instructive to look at these first.

There are three models in the WS3000 range covering a wide variety of transmission speeds and protocols from 75/1200 bps half duplex (for access to Prestel) to 1200/1200 dps full duplex transmission for user-to-user transmissons. On the front of the modem is a number of indicator lights, mostly informing the user of the current status of the device. These include the operation of the pins on the RS232 port and whether the terminal is on-line or not.

Each of the three models contains utilities that support auto-dial, auto-answer and a speed-buffered RS232 port. In addition, the WS3000 range supports the Hayes standard protocol system. This is particularly useful for business users as an increasing number of business packages include facilities that enable data to be sent via a modem directly from the program (most of these packages use Hayes protocols).

U

UART

The acronym *UART* stands for 'universal asynchronous receiver/transmitter' (see page 1600). A device that will convert serial signals into parallel ones and vice versa, it is normally incorporated into a single chip. UARTs provide an interface between the parallel data bus of a computer and an external serial link to a peripheral or communications network.

UNCONDITIONAL BRANCH

An *unconditional branch* instruction is an imperative command that instructs a program to jump to another part of the code. As its name suggests, conditions need not be fulfilled before the instruction can be performed, and the program does not have to test whether a set of conditions has been fulfilled before the branch instruction can be executed.

We can get a clear idea of the difference between 'conditional' and 'unconditional' branch instructions by comparing the two 6502 assembly language instructions BEQ and JMP. The first, 'branch if not equal to zero', is a conditional statement that requires the zero flag in the processor status register to be set before a branch can take place. The JMP ('jump') command, however, has no condition set for it, and the program will jump to the specified address irrespective of the program status.

UNIX

An operating system originally developed by Bell Laboratories for the DEC PDP-7 series of minicomputers, *Unix* was later transferred to the PDP-11. Unix is a multi-tasking system, allowing a number of users to share the same facilities, and since its introduction in 1971, has become popular among minicomputer and 16-bit micro users.

Unix was the first operating system to offer the user a flexible and powerful 'environment'. Many of its most popular features — such as hierarchical file structures — have been copied into other operating systems, such as MS-DOS. Although Unix can be called a 'standard' operating system, a number of variations have been custom-designed for particular machines or applications. Among the most popular of these are the Genix and Xenix operating systems (see our series on Unix, beginning on page 1784).

VALIDITY CHECK

A *validity check* is a piece of code that ascertains whether an item of data lies within a permitted set of parameters. Typically, a validity check is used to determine whether an input from a keyboard or other peripheral device is legal or not. For example, the BASIC statement INPUT A requires that a number, rather than an alphabetic character, be input to the computer. The computer will introduce a validity check to ensure that this is so; otherwise it will generate an error.

Programmers also incorporate validity checks within program code. These are included to prevent obviously incorrect or misleading data from being passed to the processing areas of the program. They will also prevent errors, such as those just described, from occurring that would 'crash' the program.

Generally, this is performed as an item of data enters the program. Lines of code will be added that check whether the data falls between permitted parameters. If the data fails this test, control will be passed back to the input device with a request for the data to be resubmitted.

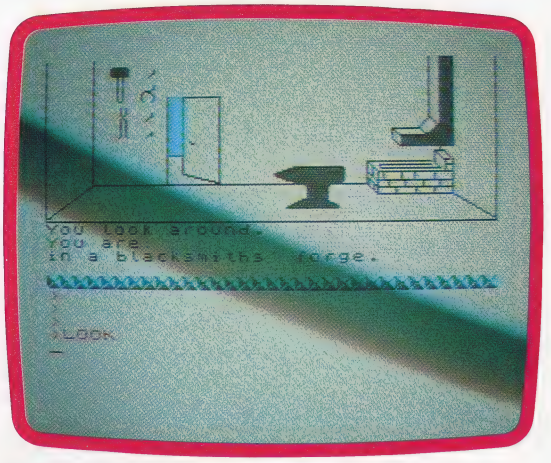
VARIABLE

A character or group of characters used within a high-level program to represent a value is called a *variable*. As the program is executed, the value of the variable may change. Variables are essential tools for manipulating data because they enable us both to assign meaningful names to data and also produce data structures. Many languages divide variables into types, such as integer, real and alphanumeric. Such categorisation helps the language's interpreter or compiler to operate more efficiently.

VDU

A *VDU*, or 'visual display unit', is an electronic device that produces a representation of alphanumeric and graphics characters on screen. By far the most common type of VDU is based around the cathode-ray tube, using 'raster scanning' techniques (see page 1440) as used on a normal television receiver. Such VDUs are either televisions or dedicated composite or RGB monitors (see page 1489). However, other forms of VDU are becoming increasingly popular on portable computers, particularly liquid crystal and plasma displays, both of which have the advantage of lower power consumption.

A VDU only provides a temporary display of information and therefore needs to be updated constantly in order to maintain the image on screen. Consequently, this allows the on-screen picture to be rapidly altered.



Refreshing Display
Most VDUs use a system whereby the electron beam scans the display every 50th of a second. The phosphor particles on the screen emit light for only a short period, and our photograph clearly shows the area of phosphor 'decay'

FREDERIC VOISIN

Wearing His Art On His Sleeve
The print-out shown here is taken from the ImageWriter. Called 'The Lion in the Jungle', the drawing was commissioned by C.S.A. records as the artwork for a record sleeve. It shows some of the range of textures and the high-resolution that can be achieved with MacPaint

MACHELANGELO

In previous instalments, we have looked at various commercial applications of graphics software such as CAD (see page 1521) and magazine page design (see page 1581). Here we look at how a French illustrator's career has undergone a change of direction as a result of discovering the MacPaint software for the Macintosh.

MacPaint, like all Macintosh software, is WIMP-based. Once the disk is booted up, the desktop offers various drawing styles. The user can choose a brush with different thicknesses, a pencil, a paintpot to fill defined areas and an airbrush can to create a stippled effect. As well as these freehand facilities, MacPaint offers the option of generating straight lines, squares, circles and other geometric shapes. Any enclosed area can then be filled with one of 38 patterns and tints — including dots, diagonal lines, lattice work and checks.

Mistakes can be erased by choosing the rubber icon, and a part of the image can be 'lassoed' and moved. The Goodies menu offers the potential for further refinements; Fat Bits, for example, enlarges an area into individual pixels, allowing the user to delete or add minute details.

In common with all Macintosh software, MacPaint is controlled by the mouse-driven cursor, which makes the keyboard unnecessary. Add-ons such as MacTablet, a small graphics tablet with a light-pen, can also be used in conjunction with MacPaint.

One dedicated convert to MacPaint is the French artist, Frederic Voisin. Eighteen months ago, he used only the traditional tools of his trade — pencils, pen, ink and paper; now, he uses the Macintosh. His first contact with a computer was on a friend's Apple Lisa. Simply drawing a line on the Lisa made him realise its potential for generating graphics. It also made him think of producing large-scale paintings rather than illustrations. This realisation was due to two characteristics of the Lisa: firstly, the fact that the image on the screen was in black and white and the obvious way to put colour onto it was to paint over the print-out; secondly, although the image was small, the high resolution of the screen immediately suggested possibilities for enlargement.

Voisin wasted no time in obtaining a Macintosh from Apple in order to experiment further. Ironically, it was a computer that directed Voisin to the classic tradition of artists. That is, pasting a rough onto the canvas and painting over it. But

Aladdin's Cave

Walking into Frederic Voisin's studio in the suburbs of Paris is like stepping into Aladdin's cave. At first glance it looks much like any other artist's work environment, with paint-spattered walls and floors, and unfinished canvases propped up on easels. The only exceptional thing about the studio is that, in a small room away from all the paint and chaos, sits a Macintosh on a drawing board.

Here, Voisin is shown with a canvas 1.5 metres high. The illustration was originally drawn on the Macintosh, then blown up to this size. The break-dancers are painted in fluorescent paint and give the impression of moving when ultra-violet light is shone on them

CLAUDIA ZEFF

instead of using a pencil to draw the rough, Voisin starts with a print out from his Imagewriter, photocopies it and enlarges it to anything up to three metres high using an architect's copier. The next part of the process is similarly traditional. Voisin pastes the copy to the canvas using a glue made of rabbits' bones — a concoction artists have used since they began painting on canvas. The glue has an elasticity that makes it possible for a large piece of paper to be stretched smoothly onto a canvas. Then he paints over the copy with acrylic, fluorescent paints to create vibrant, post-modernist paintings of robots, Zulu warriors and break-dancers. He likes to exhibit his canvasses under ultra-violet light, which combines with the fluorescent paint to create a glowing, pulsating effect similar to that of a colour VDU screen.

In the last year, Voisin produced 30 paintings on his 128 Kbyte Macintosh using MacPaint, though he maintains he has yet to discover the full potential of the program. He has enormous respect for Bill Atkinson, its designer, who, Voisin believes, has 'put himself inside the skin of illustrators' in anticipating their needs. The version of MacPaint Voisin uses is actually quite an antique, lacking many of the refinements of the more recent versions.

Cover Lines
Shown here is the original illustration used for the cover. The combination of the hard line and solid black of the faces and the softer tones of the smoke exemplifies the scope of styles MacPaint offers

COMPUTER-GENERATED ART

Unlike many of his fellow artists, Voisin is not intimidated by the idea of using a computer; he sees it as a tool to be exploited. The mouse, to him, is just like a pen — an instrument you utilise with your brain and your hand. Voisin believes that computer-generated art will eventually achieve the same status as work done in more traditional media, such as etchings. The only difference being that the original is stored on a floppy disk rather than etched onto a copper plate. Voisin thinks that this will lead to a new breed of 'media artists', producing art that is accessible to many people rather than a few art collectors. The French Ministry of Culture has made a Radiance, a large capacity computer with a colour printer, available to him. On this he has produced limited editions of colour printouts. He is looking forward to trying the colour Macintosh, although he would only use it for illustrations, and not to produce large canvasses.

Voisin has yet to experiment with the various add-ons and software, like MacDraw, available for the Macintosh. Clearly, he is no technocrat and feels he could profitably spend the rest of his life using MacPaint. One thing he is sure of is that he couldn't work without his Macintosh. 'I need it like I need the telephone or a brush and paint. The Macintosh is great. It's a revolution.'

Radiant Editions
Shown here are screen shots of Voisin's creations taken from the Radiance. The computer's high resolution and multi-colour facilities make it an ideal machine for producing quality artwork. The French Ministry of Culture has made the computer available to Voisin so that he can produce editions of colour print-outs of his work

FREDERIC VOISIN

ALL DIFFERENT TYPES

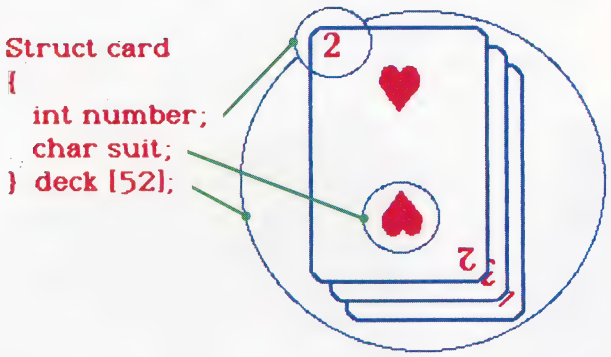
The user-defined data types in C can be improved and augmented as a means of extending this very malleable language. We look at some of them in this instalment, as well as see them working together in a bridge playing program.

As an extensible language, C can be extended to include your own libraries of definitions. We have already seen how this can be done with user-defined functions, which are simply kept on their own file and accessed by using an include in the program. You can also augment the standard user-defined data types, such as int, char and so on. The first way this can be done is to use typedef, which simply allows new names to be given for standard types. So:

```
typedef int metres;
typedef double vector[10];
typedef char *string;
```

would allow variables to be declared later in the program.

There may not seem to be much point in this, but it can serve two purposes. First, it can make programs more understandable, and with C, anything that can do this is worthwhile. Secondly, it can help overcome difficulties in moving programs between machines. The number of bytes taken up by an int, for example, is not standardised. A program that used the fact that an int was four bytes long might need a lot of work to transport to a machine that only used two bytes.



CAROLINE CLAYTON

Flexible Structure
C's struct type enables the programmer to combine different types of data within one structure. For example, a deck of cards can be represented by suit and by card number in a single structure declared as shown below

It's much easier to change a typedef at the beginning of the program rather than search through to find all occurrences in which the difference was significant. Normally, you'd want to keep all typedefs together on a separate include file.

C has similar features to PASCAL in the provision of an aggregate structure to combine associated data elements of differing types. The C struct command gives a similar structure to the PASCAL RECORD. Here is an example of its use:

```
struct card
{
   int number;
   char suit;
}
```

This defines a structure composed of two parts — an integer number and a character suit — which go together to form a playing card. Variables of this type can be declared at this and any subsequent point in the program. So:

```
struct card deck [52];
```

would form a deck of cards. The variable can be declared along with the structure definition, as in:

```
struct card
{
   int number;
   char suit;
} deck[52];
```

The tag name (card in this case) can be omitted if all variables are to be declared in one place, as in:

```
struct
{
   int number;
   char suit;
} deck[52], hands[4][13], *played;
```

but this is normally considered bad practice.

Entire structures can be assigned, passed as arguments to a function, or returned as values by a function in the same way as any of the standard functions, such as:

```
card[1][5]=deck[27];
```

or the individual elements can be referred to using a dot for a normal variable, and —> for a pointer variable of that type, as in:

```
played—>number=deck[34].number;
```

These individual elements behave just like ordinary variables of the appropriate type. A structure can be initialised in the same way as an array by following the variable name with a list of values for the elements of the structure in braces.

Structures reserve consecutive words of store for their elements; however unions use the same store for their elements, which allows you to look at memory in more than one way. For example, if a machine uses four bytes for an int, then if we wish to refer to these bytes individually, we can define a union that uses the same four bytes of store for one int or four chars:

```
union int_or_char
{
   int int_part;
   char char_part[4];
}
```

The elements of a union are referred to in the same way as the elements of a structure.

FIELDS

In normal parlance, individual elements in a structure could be referred to as fields — however, C uses this term to refer to a particular type of element that includes a bit size. We have already seen that C has bitwise operators, and fields give us the additional facility of referring by name to any group of one or more bits in a word. This is done by following the element name with a colon, followed by the bit size.

For example, in a machine with a 16-bit word and 16-bit int type, we could define:

```
struct word_in_bytes
{
    unsigned byte0 : 8, byte1 : 8;
}
struct word_in_bits
{
    unsigned bit0 : 1, bit1 : 1, bit2 : 1, bit3 : 1, bit4 : 1, bit5
    : 1, bit6 : 1 bit7 : 1, bit8 : 1, bit9 : 1, bit10 :1, bit11
    : 1, bit12 : 1, bit13 : 1, bit14 : 1, bit15 : 1;
}
union word
{
    int w;
    struct word_in_bytes w8;
    struct word_in_bits w1;
}
```

This allows us to refer to the same word of store as a whole, as two bytes or as 16-bits. Note the use of the unsigned data type, which is equivalent to int except that it allows only positive values. It is, of course, important to remember that when the size is specified in this way, that only values in an appropriate range should be assigned. An element with a bit size of one can only have the two values 0 and 1 legitimately assigned to it.

A Hand Of Bridge

The following example illustrates a few functions that might be used for a program that plays a hand of bridge. The basic idea of the game — at least as far as we are concerned here — is that the entire deck is shuffled and dealt out among four players. A feature of the hand that enables the player to decide how to play it is the point count. This counts four points for an ace, three for a king, two for a queen and one for a jack. The functions here simulate shuffling, dealing and counting points.

The card structure and the array of cards that make up the deck are defined and initialised externally so that the deck can be initialised.

```
struct card
{
    int number;
    char suit;
}
/* using minimum space for each card */
/* declare and initialise deck */
struct card hand[4][13],deck[52] =
{{1,'C'},{2,'C'},{3,'C'},{4,'C'},{5,'C'},{6,'C'},{7,'C'},
{8,'C'},{9,'C'},{10,'C'},{11,'C'},{12,'C'},{13,'C'},
{1,'D'},{2,'D'},{3,'D'},{4,'D'},{5,'D'},{6,'D'},{7,'D'},
{8,'D'},{9,'D'},{10,'D'},{11,'D'},{12,'D'},{13,'D'},
{1,'H'},{2,'H'},{3,'H'},{4,'H'},{5,'H'},{6,'H'},{7,'H'},
{8,'H'},{9,'H'},{10,'H'},{11,'H'},{12,'H'},{13,'H'},
{1,'S'},{2,'S'},{3,'S'},{4,'S'},{5,'S'},{6,'S'},{7,'S'},
{8,'S'},{9,'S'},{10,'S'},{11,'S'},{12,'S'},{13,'S'}}
shuffle(deck)
struct card deck[];
/*We assume the existence of a random number
generator — random(n) — for returning a
random number between 0 and n-1. The system
function rand() which returns a random int can be
used */
/*The cards are shuffled by swapping each card
with one other selected at random*/
{
    int i, j;
    for (i =0; i <52; ++i)
    {
        j = random(52);
        swap(&deck[i], &deck[j]);
    }
}
swap(p,q)
struct card*p, *q;
{
    struct card temp;
    temp = *p;
    *p = *q;
    *q = temp;
}
deal (deck, hand)
struct card deck[], hand[][];
{
    int i, j, k = 0;
    for (i = 0; i<4; ++i)
        for (j = 0; j <13; ++j)
            hand[i][j] = deck[k++];
}
count_points(a_hand)
struct card *a_hand;
/* a_hand is a pointer to the first card of a set of
13 in a hand, such as &hand[2][0] */
{
    int i, point_count = 0;
    for (i = 0; i < 13; ++i)
    {
        if (a_hand->number = 1)
            point_count += 4;
        else if (a_hand->number > 10)
            point_count += a_hand->number
            - 10;
        ++a_hand;
/* remember that this will now point to the next
card in deck */
    }
    return(point_count);
}
```

KNOCKED DOWN TO SIZE

Given the memory constraints of home computers, text compression can often come in handy when writing your own programs. We begin a series in which we look at the advantages of these space-saving techniques, beginning with a discussion of three methods of file compression.

Text compression techniques can be used for many different applications, of which the most obvious example for home micro users is in the programming of large adventure games. But imagine the savings in time and the reduction in phone bills that can be achieved by businesses if they compress their documents before transmitting them between offices. The effect of efficient compression techniques on piles of backup floppy disks or microdrive cartridges can be similarly dramatic. In fact, it's quite possible to achieve size reductions of between 50 and 60 per cent on reasonably large text files.

There are three basic techniques for compressing files. We will describe them separately here but in practice, they are generally used in conjunction with each other, to achieve maximum compression. The first, and simplest, method is to use a reduced character set. In most micros, a character is stored in one byte because this is the easiest size for the processor to handle. This gives eight bits and therefore 256 possible characters. When you consider that only 96 characters are needed to store both the upper and lower case alphabets, numbers and all the various punctuation symbols in the ASCII set, 256 seems rather extravagant.

The nearest number of whole bits needed to store 95 characters is seven, so coding information in seven bits instead of eight would only reduce the total size by 12.5 per cent. The obvious solution to this is to use fewer characters. Some, such as +,*,< and >, and are not usually used in document files and can be left out with reasonable safety. Even doing this, however, will not make a huge difference unless the application involved can do without the entire lower case set or cope with some equally drastic measure. The reduced character set method, therefore, has limited appeal. It is, however, used for telex communications where an extra dimension is introduced. That is, using two characters as shift characters, like the shift key on a keyboard, to switch between a set of upper case alphabetic characters and a set of numerals and punctuation symbols. This means that characters require only five bits each, giving a healthy 37.5 per cent reduction. In practice, however, the presence of shift characters reduces this.

We can actually compress text by making the character set larger. This is the second method and it involves using spare characters not used for the ASCII set as 'tokens'. Whenever one of these values is encountered in the compressed file, it is used to look up a value in a table, which contains a list of common words, parts of words or phrases, such as 'the' and 'it', or, in financial documents, 'fiscal' and so on. If these tokens are chosen carefully, large reductions can be produced — albeit at the expense of having to store a fairly large token table. Most micros, for example, store BASIC programs in tokenised form, with all the keywords stored as single bytes.

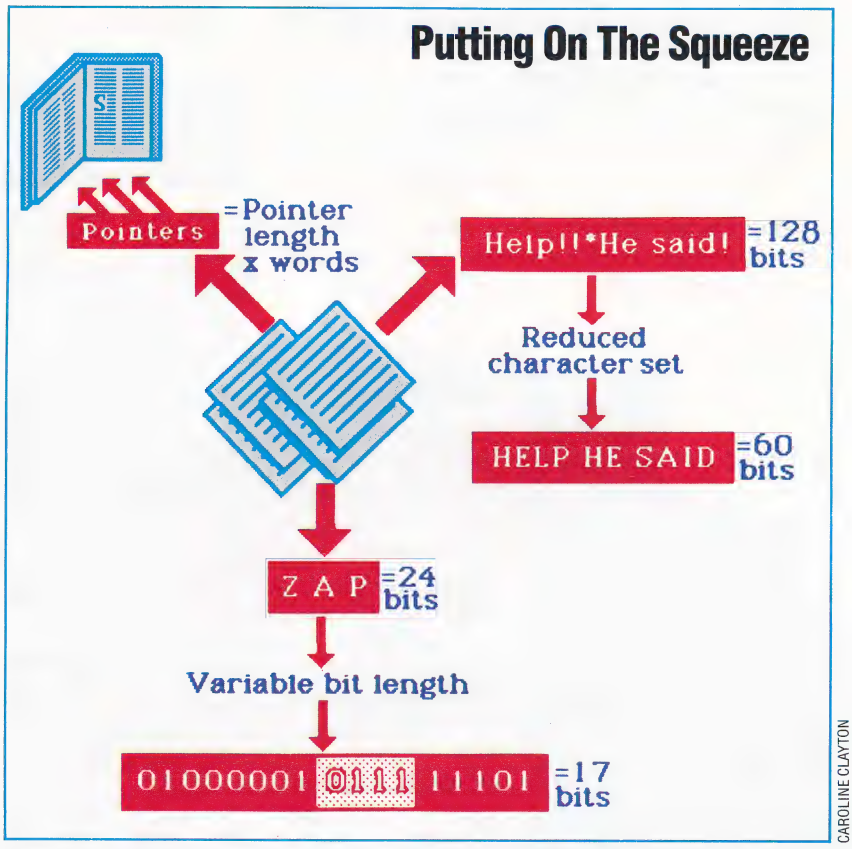
The advantages of tokenised input can be seen if your machine allows you to put a non-tokenised version of a program on disk or tape, either directly (like the Amstrad) or by LISTing to a file (as on the Spectrum) — the difference in size can be quite surprising. Although the tokens for BASIC programs are obviously fixed, the best tokenising programs for large documents will in fact search through the file to be compressed, find the optimum tokens, and store the token table along with the compressed text.

This method only works because some words

Table Of Frequencies

Many text compression programs rely on character frequencies for efficient operation. The order of frequency of alphabetic characters in the English language is as follows:

ETAONRISHDLFCMUGYPWBVKXJQZ



CAROLINE CLAYTON

Tight Fit
The inbuilt redundancy of the ASCII character set frequently necessitates some form of data compaction – either to maximise storage space or reduce data transmission times. There are three main methods of compression, which are normally combined for maximum effect. Dictionaries store commonly used words, enabling a text file to be constructed as a series of pointers. A reduced character set eliminates redundant characters (and frequently either upper or lower case as well). Finally, 'variable bit' (or Huffman) encoding assigns reduced length bit patterns to more frequently occurring letters

Cunning Compression

In order to reduce the amount of space taken up by characters, we can encode them using a technique known as 'Huffman' encoding. The first step is to calculate the relative frequency of the characters involved – so if we wished to compact the string 'ASSB', the relative frequencies of the three letters involved would be A:1, S:2 and B:1.

The next step is to encode each character, using fewer bits for the most common characters. The codes are then strung together and when encoding is required the computer simply starts at the beginning of the file and reads through it, substituting the right letters for each code. However, there's a problem here. How do we know when one code has finished and another begins? For example, if A is coded as 1, B as 0 and C as 10 – how can we tell, when we detect a sequence of 10, whether this means C or AB?

The answer is to encode each letter so that, when reading it from left to right, we can be sure that the first complete code detected – whatever its length – is the one we want. To take the A, B and C example again – we would need to code them as: 1, 00 and 01 respectively.

Our sample program, which will run on most home micros with little or no modification (Spectrum owners will need to alter the string dimension statements), does all the hard work for you. You can either see it in action as it reads from the default data provided, or you can supply your own characters and frequencies. Experiment with different values – you will find that the most dramatic bit reductions come when the relative frequencies show large fluctuation (as in the data statement provided)

```
10 INPUT "Do you wish to supply your own
 data? (y/n) ",i$
20 IF i$<>"y" AND i$<>"n" THEN GOTO 10
30 IF i$="n" GOTO 130
40 INPUT "Number of characters to encode
 ",n
50 IF n>255 THEN PRINT "Too many charact
ers...255 maximum": GOTO 40
60 DIM c$(n),f(n),r(2*n-1),t(2*n-1)
70 FOR i=1 TO n
80 PRINT "Enter character number ";i: IN
PUT s$
90 IF LEN(s$)>1 THEN PRINT "Only one cha
racter at a time...": GOTO 80
100 PRINT "Enter frequency of character
number ";i: INPUT f(i)
110 a$=a$+s$
120 NEXT i: GOTO 150
130 n=26: DIM c$(n),f(n),r(2*n-1),t(2*n-
1)
140 FOR i= 1 TO n: READ s$,f(i): a$=a$+s
$: NEXT i
150 FOR i=1 TO n: r(i)=f(i): NEXT i
160 FOR i=n+1 TO 2*n-1
170 z=9999:v=9999:k=0:g=0
180 FOR q=1 TO i-1
190 IF t(q)<>0 THEN GOTO 220
200 IF r(q)<z THEN g=k: v=z: k=q: z=r(q)
: GOTO 220
210 IF r(q)<v THEN g=q: v=r(q)
220 NEXT q
230 p=i
240 r(p)=z+v: t(k)=-p: t(g)=p
250 NEXT i
260 FOR i=1 TO n
270 c$(i)=""
280 p=i
290 IF t(p)=0 THEN GOTO 340
300 IF t(p)>0 THEN c$(i)="0"+c$(i): GOTO
 320
310 c$(i)="1"+c$(i)
320 p=ABS(t(p))
330 GOTO 290
340 NEXT i
350 FOR i=1 TO n
360 PRINT MID$(a$,i,1),r(i),c$(i)
370 NEXT i
380 END
390 DATA e,250,t,220,a,24,o,23,n,22,r,21
400 DATA i,20,s,19,h,18,d,17,l,16,f,15
410 DATA c,14,m,13,u,12,g,11,y,10,p,9
420 DATA w,8,b,7,v,6,k,5,x,4,j,3,q,2,z,1
```

and phrases are used very frequently, and it can be extended down to the individual characters themselves. This approach is the basis for the third compression technique. In both previous methods, the number of bits used for each character or token was fixed. However, variable bit length compression (also known as 'Hoffman coding' after one of its early exponents), as its name implies, requires the most common characters to be encoded into fewer bits than the less common ones.

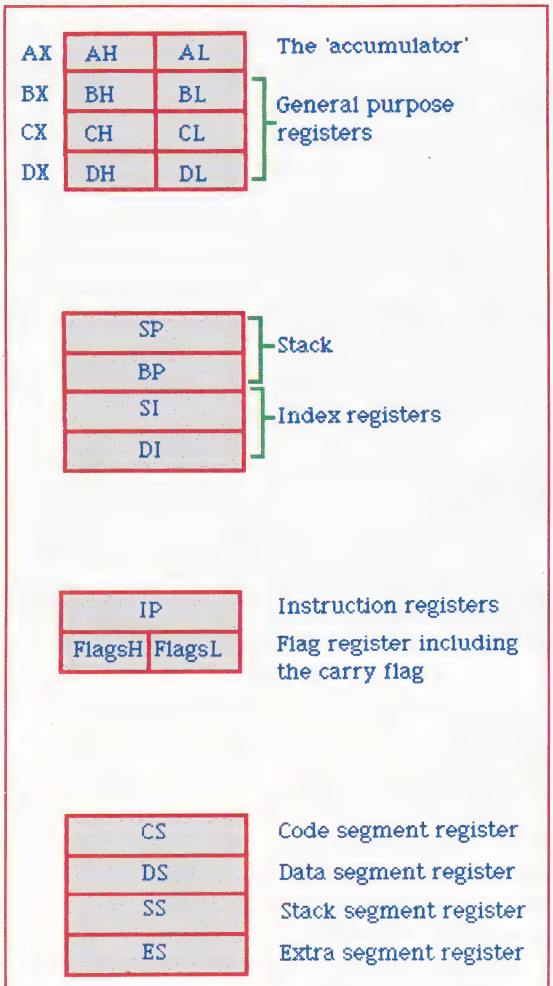
This approach isn't new — Samuel Morse used it when making up his 'dots and dashes' telegraph code, for example. Morse, however, had the advantage of extra 'characters' in the form of gaps in transmission, which he used to delimit characters and words. Even without this advantage, though, Hoffman came up with a system that relied on the fact that none of his characters could be made up of accidentally putting two other characters after each other.

This constraint means, however, that the most uncommon characters can require 17 bits or more to be stored. In order to reduce the maximum number of bits, and to add the ability to include some tokens as well, a refined method has been developed, which — although of variable length — uses only sequences of four-bit words. A four-bit word can be used to store 16 different patterns, out of which three are used as signal patterns to give information about the following four bit-elements. The other 13 are used for the 11 most used characters along with the space character and the newline character.

The three signal values are used to denote either that the next four-bit word is to be used as a character value of one of the 16 most common words, or that the next eight bits should be used to denote one of the least used characters or a less common word. This method allows the use of the full 96 characters, plus a set of 205 tokens for common words. Further refinements could include assuming that all the dictionary words start with a space (unless they are placed at the beginning of a line), thereby saving four extra bits whenever one of them is used. These features combine to give excellent results, usually with a reduction of well over 50 per cent.

GETTING IT OUT OF YOUR SYSTEM

We conclude our brief series on MS-DOS by looking at this operating system from a programmer's point of view. In particular, we demonstrate how MS-DOS function calls can be made from 8088 assembly language or from a high-level language, such as PASCAL or FORTRAN.



CAROLINE CLAYTON

Register Roll-Call

The 8086/8088 processor can address up to one megabyte of memory. This requires 20-bit addresses, and since the 8088 is based around 16-bit registers, extra bits are therefore needed to form an address. The designers of the 8086/8088 got around this problem by adopting a segment:offset format. The segment register addresses a 64-Kbyte chunk of memory and the offset register identifies the byte within the chunk. For example, the address of the current instruction is held in CS:IP.

The most important registers are A, B, C, D and the Flag register. These registers can be referred to as a complete 16-bit word using the suffix 'X' — for example AX — or as low and high bytes using the suffices 'L' and 'H'. The low byte of the D register is therefore DL

The many system calls available in MS-DOS fall broadly into six different categories:

- Character I/O from and to the standard system devices.
- File handling (including directory management).
- Memory management.
- Process management.
- Miscellaneous system 'function' calls.
- Special calls for Microsoft Networks.

All these system services may be called from any application, and they provide a unified and comprehensive interface that ensures compatibility between different operating system versions. Any calls to absolute addresses or direct to hardware devices will not necessarily be portable to other MS-DOS systems or, indeed, compatible with different DOS versions — even from the same manufacturer.

The use of the word 'function' to describe all system calls is standard MS-DOS terminology, derived from BCPL and C via Unix. It does *not* mean that the service is just a function that returns some data. It also includes routines that actually do something useful (such as opening files and so on). These are procedures, of course, but are nevertheless loosely termed functions in all the literature.

The most useful general-purpose functions are the 'miscellaneous' calls. Each is affected by a software interrupt and will always entail performing the following steps:

1. Move any data required into the appropriate registers, (a drive number, for example, may be required in DL).
2. Put the function number in AH (the high byte of AX).
3. Issue the software interrupt 21H.

Any data returned by the function will be found in the 8086 registers on return from the call, either directly or via a pointer or address. So, for example, the DOS function 19H (all function numbers are in hexadecimal) returns the disk drive that is currently selected as a code in the AL register. It requires no action code, so it may be simply coded as:

```
mov     ah, 19H     ; get selected drive
int     21H         ; call the fuction
```

The result is returned in the AL register, and is a number representing the drive (A = 0, B = 1 and so on). By adding the ASCII code for the character 'A' to this result and calling the DOS function 05H (print the code in DL as an ASCII

character), we could display the logged drive as an upper case character:

```
add al, 'A'          ; convert 0 to 'A', 1 to 'B' etc
mov dl, al           ; move it to the DL register
mov ah, 05H          ; print character
int 21H              ; call the function
```

A Complete Function

To round off our look at MS-DOS programming, we present a complete high-level function call. It is very often necessary to find out how much disk space is available before writing data to a file. Attempting this when the disk is nearly full can be disastrous! By calling the DOS function 36H (54 decimal) we can check the disk before we write to it and, if necessary, give ourselves the option of closing files or swapping disks if required.

To execute this system call, the function number is placed in AH as usual, and DL must contain the drive number; drive A is 1, B is 2 and so on. A code of 0 indicates the default drive. All we need to do is initialise the DL field (mapped to the low byte of the D register) to the required value, put 36H into AH and make the system call. On return, AX will contain the error code OFFFFH if all is not well (an invalid drive code was given, for instance), otherwise the following data (showing what each register holds) is returned:

AX	The number of sectors per cluster
BX	The clusters available
CX	The number of bytes per sector
DX	The total number of clusters

Thus the total disk space available (in bytes) is the product of the values:

clusters × sectors per cluster × bytes per sector

Note that a cluster is a unit of disk space allocation, and corresponds to a number of complete sectors.

```
FUNCTION DiskSpace ( drive : integer ) : integer;
  { returns the number of bytes free on drive }
  VAR
        registers : SysReg; {see manual}
  BEGIN
    WITH registers DO
      BEGIN
        DL := drive; { 0 = default, 1 = A, etc}
        AH := 36H; { Function number }
        system ( registers ); { call it }

        IF AX = OFFFFH
          THEN { there was an error, so : }
            BEGIN
            WriteLn ( 'DiskSpace : ERROR
            (invalid code ?)' );
            DiskSpace := 0 { for all we know ! }
          END
        ELSE
            DiskSpace := AX * BX * CX
      END { sectors * clusters * bytes }

END: { DiskSpace`
```

The mnemonics used in the previous examples are for Microsoft's own assembler, but this is not always supplied to the end user by the OEM. Fortunately, it is not essential to have an assembler — as we'll see, calling MS-DOS from high-level languages is usually very easy. If you intend to do a lot of machine code programming, however, make sure that you purchase an assembler that generates relocatable machine code files in standard Intel (.OBJ) format. This is the *de facto* standard for 8086 family machines, and most linkers (including those for PASCAL, FORTRAN, C and so on) from reputable suppliers will enable you to link editing modules written in any of these languages (with assembled machine code libraries if desired).

HIGH-LEVEL PROGRAMMING

One alternative to using an assembler is the BASIC standby of POKEing 'manually assembled' instructions into a convenient block of spare memory. This is only to be recommended, however, for fairly trivial tasks such as the short routines just illustrated. You will need an Intel op-code list, a detailed memory map of your machine, knowledge of the transient program space and a fair bit of patience! Especially from the point of view of readability, this option is best avoided. Fortunately, most compiled language implementations for MS-DOS include the necessary extensions to call the system services without the need for writing external machine code and linking to the main program.

Microsoft's own PASCAL-86 has a procedure called DOSXQQ, giving direct 'function' requests. Digital Research's MT+86 has a similar routine. Perhaps the best approach (and certainly the simplest) is that adopted by both Prospero Software's well-regarded ISO-validated compiler (Pro PASCAL) and the inexpensive Turbo PASCAL. These offer an excellent vehicle for DOS programming without the need to use assembly language.

Both Pro PASCAL and Turbo PASCAL have a TYPE descriptor that maps all the essential 8086 registers on to a RECORD, the fields of which can be initialised before a call and accessed afterwards to return data from the DOS function. The syntax is similar in each case (refer to the PASCAL manual for complete details).

Prospero has a procedure called, logically, system, which takes a variable parameter of the type mentioned (SysReg) and performs the actual function call. So, the complete 'print the currently selected drive' procedure just given in assembler mnemonics would become:

```
WITH registers DO
  BEGIN { print the selected drive : }
    AH := 25; { Function code 19H }
    system ( registers ); { call it }
    WriteLn ( 'Currently selected drive is ',
      chr ( AL + ord ( 'A') ), ':' )
  END
```

STACKS OF IDEAS

In the previous instalment, we looked at the use of subroutines and noted that this form of program control is particularly well-suited to structured programming techniques. We examine now the subroutine and its relationship to the stack in a little more detail before moving on to discuss parameter passing.

To begin, let's look at an example that makes two calls to the subroutine CALC, which we discussed in the previous instalment (see page 1818).

```
                    . . . .          *lines of code
2000  4EB8          JSR  CALC  *call the
                                 subroutine
                    (. . . .lines of code. . . .)
2100  4EB8          JSR  CALC  *and again
2102  4000
                    (. . . . lines of code . . . .)
4000  3200    CALC  MOVE D0,D7 *subroutine entry
                    (. . . . lines of code . . . .)
4100  4E75          RTS        *return to call
```

Each time the JSR instruction is executed, the 68000 pushes the contents of the program counter (the address of the next instruction following the JSR instruction) onto the stack and loads the address of the routine specified by JSR into the PC. The subroutine is then executed, and when an RTS is encountered the 68000 pops the return address (known as the *linkage address*) off the stack into the PC. Note that the linkage address is saved in full long word format on the stack, so in the example — after the first CALC has been executed — the stack and PC hold values as shown at the top of the Linkage Address diagram.

After the RTS instruction has been executed at the end of the CALC subroutine, the linkage address will have been restored in the PC and the stack will be as shown in the second half of the diagram. Notice that the linkage address still remains written on the stack — it will just be overwritten when the stack is next used.

This linkage mechanism will have been taken care of by the 68000 hardware. The significance of this, however, is that if we nest subroutines, the linkage mechanism will automatically take care of that situation, as well.

Clearly, this method of stacking linkage addresses automatically takes care of nested subroutines by expanding the stack. Let's suppose, however, that CALC calls itself (a recursive call). Again, the linkage mechanism will take care of the situation by stacking all the linkage addresses on top of one another until the recursion ends, or until disaster strikes in the form of some addressing violation — that is, our stack has grown too big!

Let's return now to the problem of parameter passing, for both input and output. In the last instalment, we passed data into the subroutine CALC using D1 and recovered data from it with D2. This simple solution to the problem is satisfactory for small programs, but for larger programs we need a more general solution, especially since there are only a finite number of registers.

One method used to overcome this problem, particularly with compilers, is to use the stack as a means of parameter passing. To pass parameters to CALC, for example, we could use:

```
2000   MOVE PARAM1,-(SP)  *put first
                           parameter on stack
2004   MOVE PARAM2,-(SP)  *and second
2008   JSR CALC           *call CALC
200A   . . . .
```

where the stack on entry to CALC would hold:

```
                PARAM1
                PARAM2
                ls link
SP in CALC →    ms link
```

In order to access PARAM1 we would therefore have to use a displacement on the SP in order to 'jump over' the linkage address:

```
ADDQ  #6,SP     *adjust pointer
MOVE  (SP),D4   *and pick up
                 parameter
```

or, more simply:

```
MOVE  6(SP),D4
```

Of course, if we adjust the stack pointer in any way we must make sure that the SP points to the linkage address before the RTS instruction is executed — otherwise, disastrous and unpredictable results occur. This principle also applies if CALC passes parameters back via the stack — unless, of course, the subroutine overwrites one or both of the input parameters.

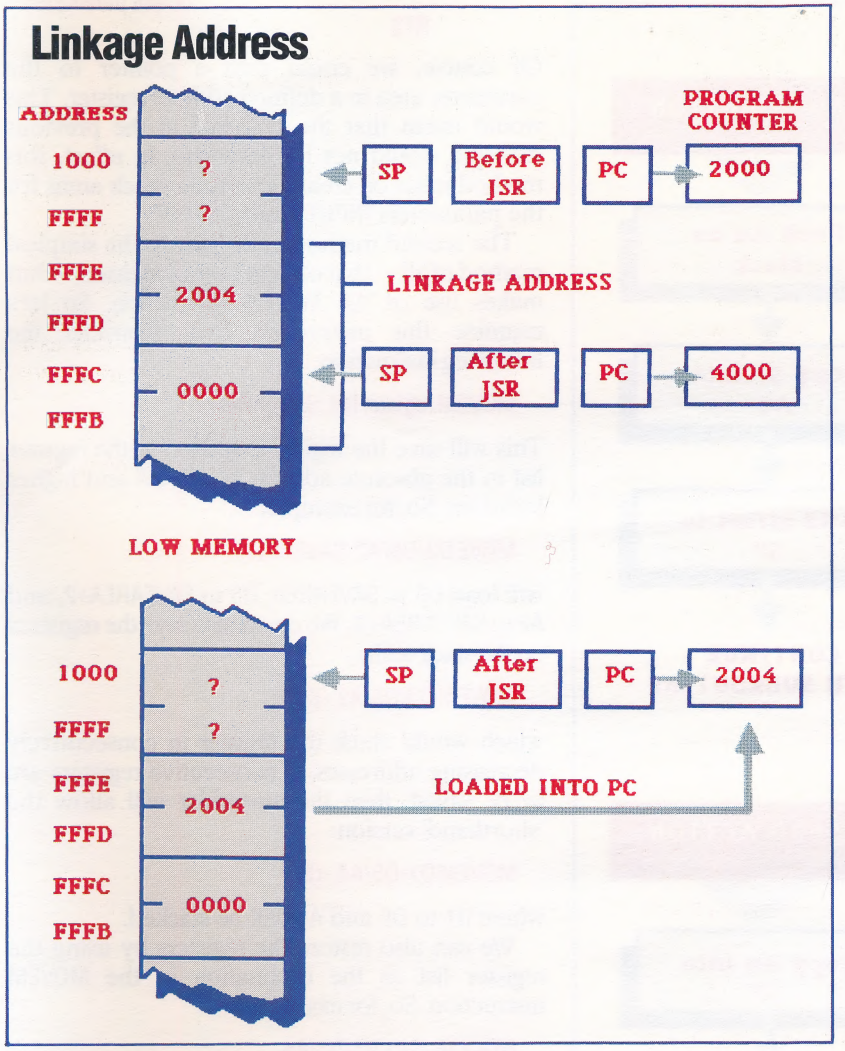
However, an even simpler method would be to use a separate stack for the parameters, say A6, and leave the stack pointer alone (reserved for hardware machine functions). In this case, the call would become:

```
MOVE  PARAM1,-(A6)     *set first
                        parameter on A6
                        stack
MOVE  PARAM2,-(A6)     *and second
JSR   CALC             *keep SP for
                        linkage
```

and within the subroutine we could easily pick up the parameters using, for example:

```
MOVE  (A6)+,D2         *get second
                        parameter
MOVE  (A6)+,D1         *and first
```

since there is no calling routine linkage to 'step over'. Notice also that the parameters will be



Unbroken Link
The 68000 saves a return address (the 'linkage address') on the stack before diverting program control to a subroutine. This address is saved in long-word format on the stack and is restored to the Program Counter when an RTS instruction is encountered

picked up in reverse order compared to the call.

Stacks are used in modern high-level block-structured languages like PASCAL in a very ordered, structured fashion. The typical stack structure will allow parameters and local variables of a procedure (which will be implemented in the executable code as a subroutine) to be allocated space on the stack within a defined area. So, for example:

```
Procedure Calculate (xval, yval:int);
var
        store:int;
begin
        store:= 2*xval + yval^2;
```

as a fragment of PASCAL would have the input parameters xval and yval, and the local variable store, allocated space in the stack. In addition, any unnamed temporary storage that the compiler might require (say, to hold the yval^2 component of store while the 2*xval is evaluated) would use the stack in the normal push and pop way.

This arrangement is well-structured from the compiler viewpoint, but on some computers it could lead to a fairly large overhead in manipulating the stack data. On the 68000, the situation is considerably eased by the use of two instructions: LNK for link, and UNLNK for unlink.

These instructions are used together and allow easy manipulation of data by reserving blocks of memory within the stack for subroutine usage. Following a subroutine entry, LINK sets up a defined address register (called the *frame pointer*, FP) to a data area on the stack, and moves SP down the stack by a defined amount of displacement. For example, if the stack was:

```
        param1
        param2
        ls linkage
SP  →   ms linkage
```

after the JSR had been executed, then the state after a LNK instruction is executed would be:

```
                param1
                param2
                ls linkage
                ms linkage
          FP →  old FP
           |    named
displacement    local
           |    variables
          SP
```

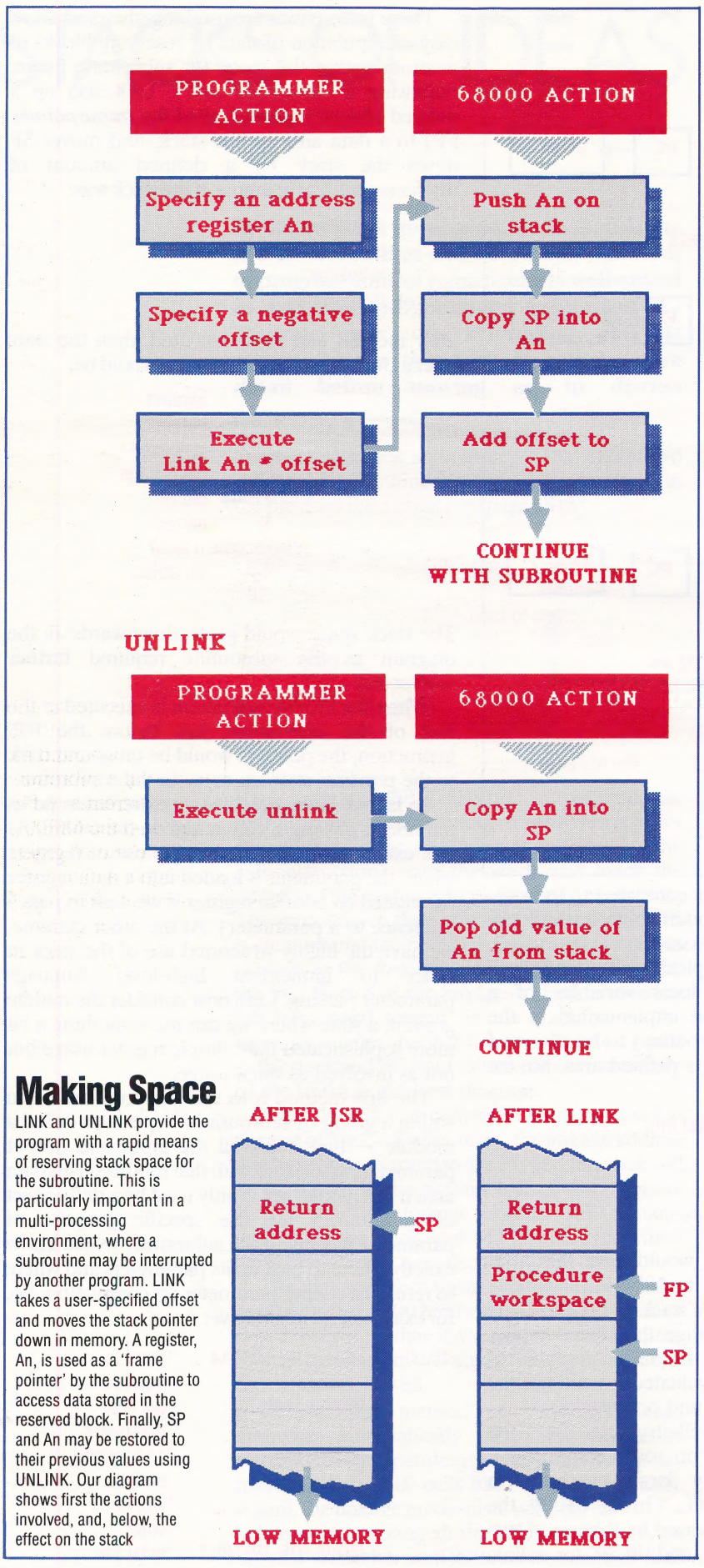
The stack space would grow 'downwards' in the diagram as the subroutine required further workspace.

When the UNLNK instruction is executed at the end of the subroutine, just before the RTS instruction, the pointers would be unwound back to the previous state on entry to the subroutine.

So far, we have seen the two extremes as far as parameter passing is concerned on the 68000. At one extreme we have the simple use of registers where the parameter is loaded into a data register (or indeed an address register if we wish to pass a reference to a parameter). At the other extreme, we have the highly structured use of the stack in order to implement high-level language parameter passing. Let's now consider the middle ground, a state where we can use something a bit more sophisticated than simple register usage but not as involved as stack usage.

The first method is to use a defined data area within a group of subroutines — perhaps called a module — that holds all the input and output parameters associated with that module. This data area is not global, but is only used by a defined set of subroutines for the specific purpose of parameter passing. Each subroutine would know exactly where to pick up its parameters and where to return its output parameter on completion. So, for example, we might have:

```
CALC    LEA   INPUTS,A4     *point to
                             input param area
        MOVE  (A4),D0       *get first
                             parameter
        MOVE  (A4)+,D1      *and second
                             one too
        ....                *lots of
                             code
        MOVE  D5, OUTPUT    *enter the
```



Making Space

LINK and UNLINK provide the program with a rapid means of reserving stack space for the subroutine. This is particularly important in a multi-processing environment, where a subroutine may be interrupted by another program. LINK takes a user-specified offset and moves the stack pointer down in memory. A register, An, is used as a 'frame pointer' by the subroutine to access data stored in the reserved block. Finally, SP and An may be restored to their previous values using UNLINK. Our diagram shows first the actions involved, and, below, the effect on the stack

CAROLINE CLAYTON

```
                                    output parameter
        RTS
```

Of course, we could pass a pointer to the parameter area in a defined address register. This would mean that the first line in the previous example would not be necessary. In effect, this method relies on creating separate stack areas for the parameters INPUTS and OUTPUT.

The second method goes back to the simplest method of all — that of direct use of registers — but makes use of the MOVEM instruction. So let's examine this instruction first. Consider the following instruction:

```
MOVEM register list, SAVEAREA
```

This will save the registers defined in the register list in the absolute address SAVEAREA and higher locations. So, for example:

```
MOVE D3/D5/A2, SAVEAREA
```

will load D3 to SAVEAREA, D5 to SAVEAREA+2, and A2 to SAVEAREA+4. We could also save the registers on the stack with:

```
MOVEM D3/D5/A2,-(SP)
```

which would stack the register in consecutively decreasing addresses. If consecutive registers are to be saved, then the assembler will allow the 'shorthand' version:

```
MOVEM D1-D5/A4,-(SP)
```

where D1 to D5 and A4 will be stacked.

We can also restore the registers by using the register list as the destination in the MOVEM instruction. So, for example:

```
MOVEM-(SP),D1-D5/A4
```

would restore the registers stored in the previous example.

We must ask ourselves how this helps us in our consideration of parameter passing. The answer lies in the convenience of storing registers that have been allocated to a specific purpose, say system use, and then being free to use those registers as we wish. For example, if we allocated D0 and D1 as parameter registers, then provided we stored the remaining registers on entry to a subroutine, we could use the remaining registers as we required in the subroutine. For instance:

```
CALC    MOVEM  D2-D7,-(SP)    *save old
                               D2-D7
        MOVE   D0,D2          *and load
                               first parameter
        .....                 *lots of
                               code using
                               D2-D7
        MOVEM  -(SP),D2-D7    *and restore
                               old values
        RTS
```

From this you can see that the 68000 allows us to use several methods to pass parameters to subroutines and also provides some useful tools to do it!

HAVE YOU ORDERED YOUR VOLUME EIGHT BINDER YET?

FOLD 2

IF YOU HAVE NOT ASKED FOR THE BINDERS TO BE SENT TO YOU AUTOMATICALLY, DO SO NOW, AND ENSURE THAT YOUR COPIES OF THE HOME COMPUTER ADVANCED COURSE ARE KEPT PROPERLY BOUND AND IN GOOD CONDITION FOR YEARS TO COME.

BY TICKING THE BOX OPPOSITE, YOU WILL BE SENT BINDER NUMBER 8.

☐ PLEASE SEND ME MY VOLUME 8 BINDER NOW. I ENCLOSE A CHEQUE/POSTAL ORDER FOR £3.95 (WHICH INCLUDES POSTAGE AND PACKING).

FOLD 3

BY TICKING THE OTHER BOX *AS WELL*, YOU WILL BE SENT SUBSEQUENT BINDERS FOR YOUR COLLECTION.

☐ I WOULD ALSO LIKE TO RECEIVE FUTURE BINDERS AS THEY ARE ISSUED. I UNDERSTAND THAT I WILL RECEIVE A PAYMENT ADVICE FOR £3.95 (WHICH INCLUDES POSTAGE AND PACKING) WITH EACH BINDER. IF I AM NOT SATISFIED WITH THE BINDER, I CAN RETURN IT TO YOU, WITHIN 14 DAYS, AND OWE NOTHING.

NO STAMP NECESSARY.

JUST FOLD UP THE PAGE AS INDICATED, REMEMBERING TO ENCLOSE YOUR CHEQUE/POSTAL ORDER MADE PAYABLE TO ORBIS PUBLISHING, AND SEND TO US TODAY.

FOLD 4

FOLD 4

I enclose a cheque/postal order made payable to: Orbis Publishing Ltd. for a total of £____ which I understand includes the cost of postage and packing.

NB: Please allow 28 days for the delivery of your binders.

When you have completed the order form fill in your name and address in the space provided.

Then cut along the dotted line to detach the page, enclose your cheque/postal order, and fold the page carefully – following the instructions to complete the reply paid envelope.

NO STAMP NECESSARY.

FOLD 1

IF YOU ALREADY HAVE AN ACCOUNT NO. FOR YOUR BINDERS PLEASE FILL IT IN HERE.

Complete the section below with one letter or figure per space.

MR		INITIALS	SURNAME
MRS			
MISS			

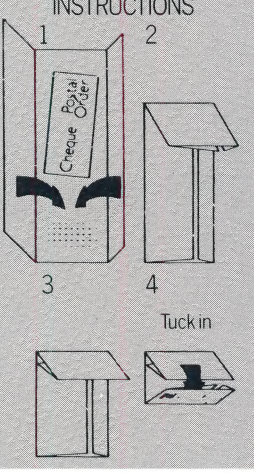
ADDRESS & POST CODE

TELEPHONE NO. INCLUDING STD CODE OR EXCHANGE NAME

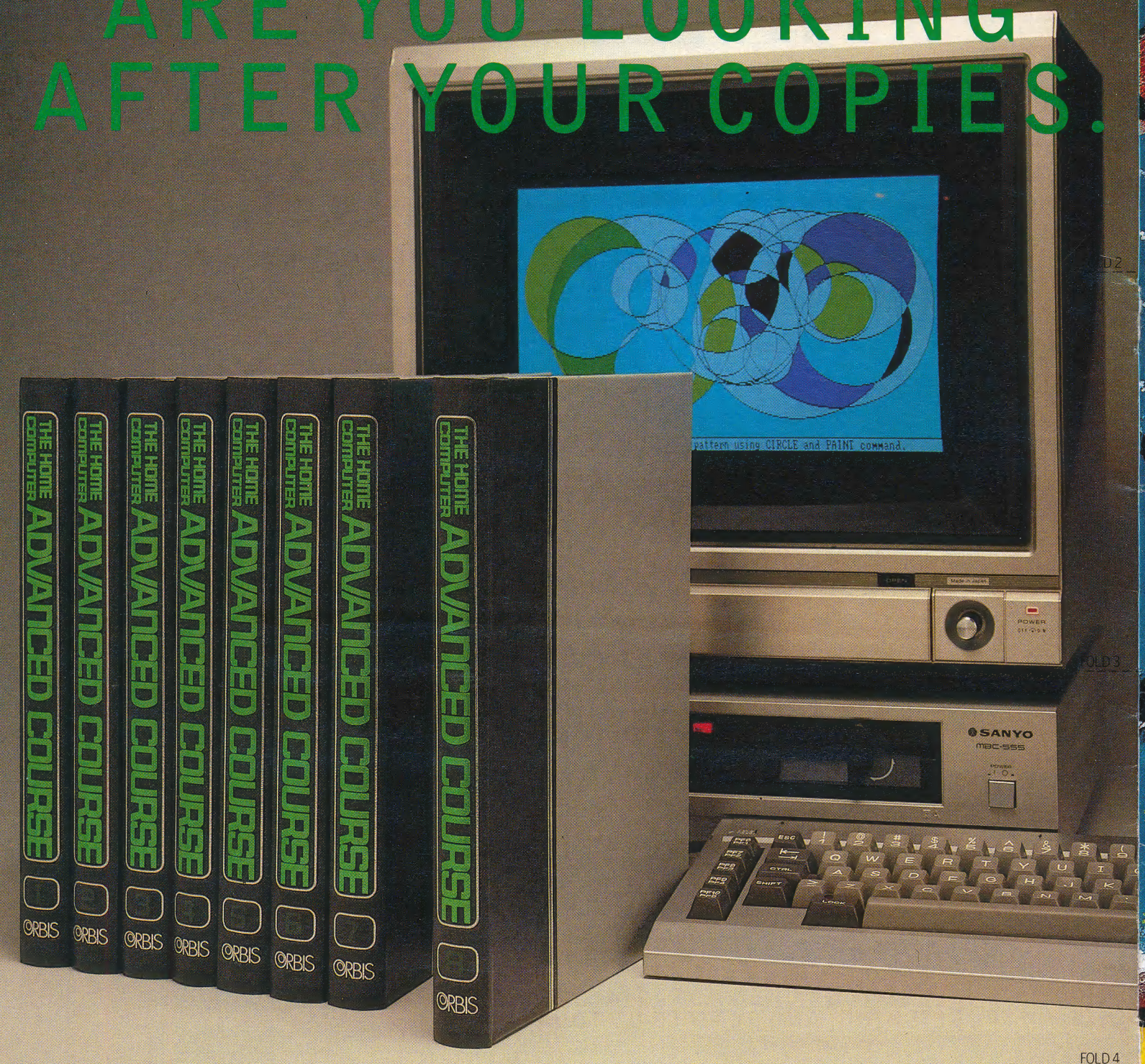
FOLD 1

FOLD 4

FOLDING INSTRUCTIONS



ARE YOU LOOKING AFTER YOUR COPIES.

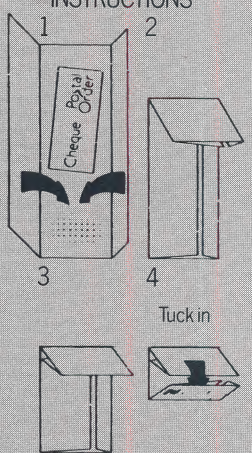


FOLD 2

FOLD 3

FOLD 4

FOLD 4
FOLDING INSTRUCTIONS



FOLD 1

Postage will be paid by licensee

Do not affix Postage Stamps if posted in Gt. Britain, Channel Islands or N. Ireland.

BUSINESS REPLY SERVICE
Licence No. SW4035.

The Home Computer
Advanced Course Binders
Orbis House, 20-22 Bedfordbury
London WC2N 4BR

2

FOLD 1

GUARANTEE GUARANTEE GUARANTEE GUARANTEE GUARANTEE

GUARANTEE

If you are not entirely satisfied with your binder, send it back immediately and it will be either exchanged, or, if you prefer, your money will be refunded in full.

GUARANTEE